Generative Adversarial Nets on MNIST and SVHN

Jiefu Zhou, Liangcao Ling, Saier Gong, Weichen Li
jz3148, ll3337, sg3772, wl2726
Department of Statistics
Columbia University
New York, NY 10027

Abstract

In this paper, we implement two different Generative Adversarial Nets (DCGAN and WGAN), then train them on both MNIST and SVHN datasets. We evaluate GANs by tuning different sets of hyper-parameters and comparing generated images with real images. In the end, we find that both DCGAN and WGAN can have a relatively good as the number of epoches increases if the loss of generator and discriminator are close, and we need deeper and more powerful networks with multiple Convolutional layers to generate more complex images in the real world.

1 Introduction

Generative adversarial networks [1] (GANs) a class of methods for learning generative models based on game theory. GANs are able to learn how to model the input distribution by training two competing (and cooperating) networks referred to as generator G and discriminator D (sometimes known as critic). The role of the generator is to keep on figuring out how to generate fake data or signals (this includes audio and images) that can fool the discriminator. Meanwhile, the discriminator is trained to distinguish between fake and real signals. As the training progresses, the discriminator will no longer be able to see the difference between the synthetically generated data and the real data. From there, the discriminator can be discarded, and the generator can then be used to create new realistic data that have never been observed before. However, GAN also has some problems, for example, as the discriminator gets better, the gradients of generator vanish, and GAN training is massively unstable, etc.[14] With the use of Earth-Mover(EM) distance or Wasserstein-1 distance[7], here comes Wasserstein Gan, and gradients vanishing and unstable training issues have all been solved.

1.1 Previous Work

Several recent papers focus on improving the stability of training and the resulting perceptual quality of GAN samples [2, 3, 5, 6, 7]. We build on some of these techniques in this work. For instance, we use some of the "DCGAN" architectural innovations proposed in Radford et al. [3], as discussed below. And we also borrowed some ideas from online sourses (tf tutorial [8], Naoki's blog [9]). Besides, WGAN has been proposed to improve the original GAN[7].

1.2 Goal of our work

One of the primary goals of this work is to improve the effectiveness of generative adversarial networks for semi-supervised learning (improving the performance of a supervised task, in this case, classification, by learning on additional unlabeled examples). Like many deep generative models, GANs have previously been applied to semi-supervised learning [10, 11], and our work can be seen as a continuation and refinement of this effort. Overall, in this work, we implemented three GANs (Valina GAN, DCGAN, and WGAN) on two datasets (MNIST and SVHN).

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

1.3 Data

We performed experiments on MNIST and SVHN (format 2).

1.3.1 MNIST

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. The input dimension is 4, and the input shape is (60000, 28, 28, 1). The output dimension is 1.

1.3.2 SVHN

SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. The SVHN (format 2) database has a training set of 225045504 examples, and a test set of 79970304 examples. The input dimension is 4, and the input shape is (73257, 32, 32, 3). The output dimension is 1.

2 Methods

We start from generative models and generative machine learning is training a model to learn parameters maximizing the joint probability of $P(X, Y)$ in a supervised learning. However, generative model requires more data than discriminator. It has difficulty approximating many intractable probabilistic computations that arise in maximum likelihood estimation and related strategies, and leveraging the benefits of piecewise linear units in the generative context [1]. To solve this problem, GANs improved generative models by adding a discriminator.

GANs is a Two Player Game [12]. It includes the two models, the generator and the discriminator.

- The generator **G** generates a batch of samples from noises, and these, along with real examples from the domain, are provided to the discriminator and classified as real or fake [13]. Generator's training objective is to decrease the error rate of the discriminator network.
- The discriminator **D** tries to classify if an input image is real or generated. Discriminator's training objective is to increase the error rate of the discriminator network.

Competition in this game drives both teams to improve their methods until the counterfeits are indistinguishable from the genuine articles.

The whole process can be explained by the following formula:

$$\min_G \max_D L(D, G) = E_{x \sim p_{data}(x)}[logD(x)] + E_{z \sim p_z(z)}[log(1 - D(G(z)))] \tag{1}$$

where,

- G = Generator
- D = Discriminator
- $p_{data}(x)$ = distribution of real data
- $p_z(z)$ = distribution of generator
- x = sample from $p_{data}(x)$
- z = sample from $p_z(z)$
- D(x) = Discriminator Network
- G(z) = Generator Network

However, GAN also faces some issues like gradients vanishing and unstable training, therefore, WGAN is used. It keeps most algorithms in the original GAN, but only changes the following four points:

- In the Generator G network, in the last Dense layer, we no longer use sigmoid as activation function.

- Unlike the loss function in original GAN, the loss of WGAN does not need to do log operation.

$$DiscriminatorLoss = E_{z\sim p_z(z)}(D(G(z))) - E_{x\sim p_{data}(x)}(D(x))$$

$$GeneratorLoss = -E_{z\sim p_z(z)}(D(G(z)))$$

- Every time after undating the Discriminator weights, we keep the value of weight in $[-c, c]$.
- RMSprop optimizer method is preferred.

3 Architecture

Since MNIST only has one color channel, we just reshape each image to the dimension of $[1, 28 * 28]$. Therefore we will not use CNN in discriminator, just Dense layer together with activation function are enough. Here is architecture of Valina GAN and WGAN on MNIST dataset:

Generator

- First Layer: Dense
  - Input: 100×1 vector
  - Units: 128
  - Activation: LeakyRelu($\alpha = 0.01$)
- Second Layer: Dense
  - Input: 128×1 vector
  - Units: 784, which is equal to 28×28
  - Activation: Tanh

Discriminator

- First Layer: Dense
  - Input: 784×1 vector
  - Units: 128
  - Activation: LeakyRelu($\alpha = 0.01$)
- Second Layer: Dense
  - Input: 128×1 vector
  - Units: 1
  - Activation: Softmax(in Valina GAN)
  - Activation: Linear(in WGAN)

Since SVHN has three color channels, We need a deeper and more powerful network than the Valina GAN. This is accomplished through using convolutional layers in the discriminator and generator. It's also necessary to use batch normalization to get the convolutional networks to train. So we will use Deep Convolutional Generative Adversarial Nets (DCGAN), which means we uses convolutional and convolutional-transpose layers in the discriminator and generator, respectively. We choose CNN because of neural network's ability to generate data (e.g., upsampling feature maps to create new images). The discriminator is often built using a regular CNN because of its ability to break data (e.g., images) down into feature maps, and to ultimately classify data (e.g., to determine if an image is real or fabricated).

- In generator, we will try convolutional-transpose , batch normalization, dropout, LeakyReLU activation and tanh activation.
- In discriminator, we will try convolution , batch normalization, dropout and LeakyReLU activation.

AS for Wasserstein Generative Adversarial Nets (WGAN), we use the same structure except for deleting sigmoid in the last layer of Discriminator. Here is architecture of DCGAN and WGAN on SVNH dataset:

Generator

- First Layer: Dense
  - Input: 100×1 vector
  - Units: $4 \times 4 \times 512 = 8192$
  - Batch Normalization
  - Activation: Leaky Relu
  - Reshape: 8192 to $(4, 4, 512)$
- Second Layer: Conv2DTranspose
  - Input: $(4, 4, 512)$
  - Filter: 256
  - Kernel Size: 5
  - Stride: 2
  - Padding: same
  - Batch Normalization
  - Activation: Leaky Relu
- Third Layer: Conv2DTranspose
  - Input: $(8, 8, 256)$
  - Filter: 128
  - Kernel Size: 5
  - Stride: 2
  - Padding: same
  - Batch Normalization
  - Activation: Leaky Relu
- Fourth Layer: Conv2DTranspose
  - Input: $(16, 16, 128)$
  - Filter: 3
  - Kernel Size: 5
  - Stride: 2
  - Padding: same
  - Batch Normalization
  - Activation: tanh

Discriminator

- First Layer: Conv2D
  - Input: $(32, 32, 3)$
  - Filter: 64
  - Kernel Size: 5
  - Stride: 2
  - Padding: same
  - Activation: Leaky Relu
- Second Layer: Conv2D
  - Input: $(16, 16, 64)$
  - Filter: 128
  - Kernel Size: 5
  - Stride: 2
  - Padding: same
  - Batch Normalization
  - Activation: Leaky Relu
- Third Layer: Conv2D
  - Input: $(8, 8, 128)$
  - Filter: 256
  - Kernel Size: 5
  - Stride: 2
  - Padding: same
  - Batch Normalization
  - Activation: Leaky Relu
- Fourth Layer: Flatten
- Fifth Layer: Dense
  - with the number of output unit 1
  - Activation: sigmoid (in DCGAN)

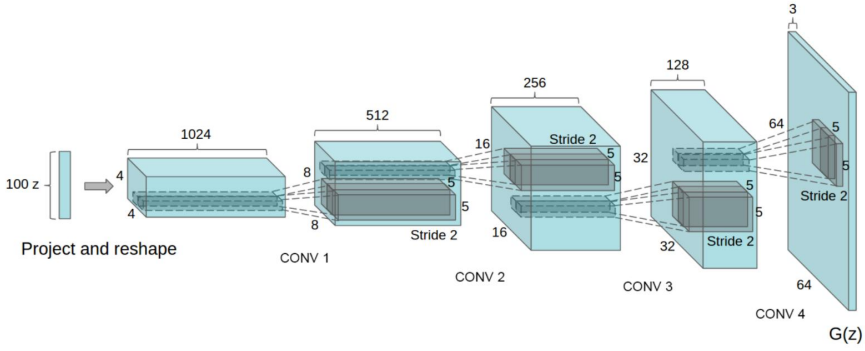

Figure 1: Architecture on SVNH dataset

4 Conclusions

When training the WGAN on MNIST dataset, we use rmsprop with learning rate 0.00005 as the optimizer for both discriminator and wgan training process, and 0.01 as the constance to limit the value of weights in discriminator. Since we use a simple architecture on MNIST, it won't take a long time running the process on GPU. Therefore, we set the number of training epochs to 50000. From **Figure 5**, we can see, that at the beginning, D loss and G loss are both big, but as the training epoch increases, D loss and G loss seem to be stable, around zero. From **Figure 6**, we can see the generated images from WGAN are much clearer than the results from GAN in **Figure 2**. However, due to the hyperparameter tuning, WGAN needs to train more epochs than GAN to achieve clear generated images.

According to the results of generating SVHN images, we find that both DCGAN and WGAN can have a relatively good result in 30 epochs. However, we need to try many different sets of hyper-parameters to get a good model for DCGAN. As we take a closer look at the plot of the training losses, we see that the loss of the generator are bigger than the loss of the discriminator most of the time. This indicates that we need to set a smaller learning rate for generator and a bigger learning rate for discriminator in order to have a stable training process, otherwise the loss of the generator may increase and decrease sharply between epochs.

Sometimes GAN failed to learn while the WGAN still was able to produce samples. Even though we can add more layers in DCGAN, it's more stable to use WGAN. Still the performance of WGAN can be improved by some adjustment like using WGAN-GP. In WGAN, we use weight clipping, while gradient penalty is a more efficient way.

4.1 MNIST

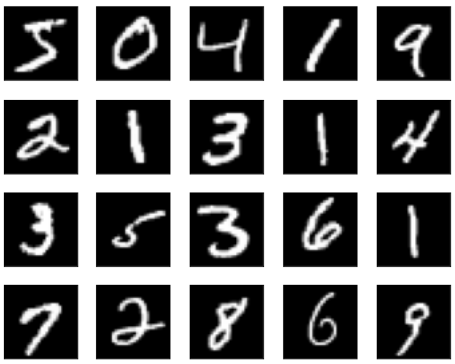

Figure 2: Real Data on MNIST

4.1.1 Valina GAN on MNIST

Figure 3: Training Losses of Valina GAN

Figure 4: Generated images using Valina GAN

4.1.2 WGAN on MNIST

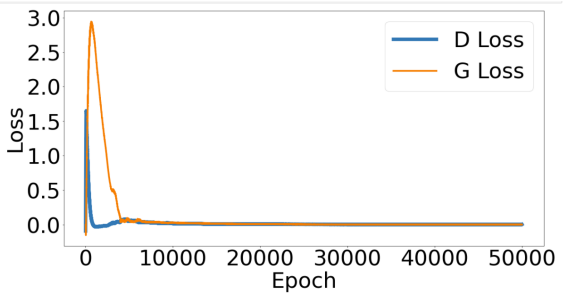

Figure 5: Training Losses of WGAN

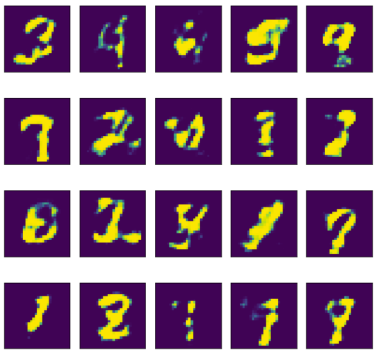

Figure 6: Generated Images using WGAN

4.2 SVHN

Figure 7: Real Data on SVHN

4.2.1 DCGAN on SVHN

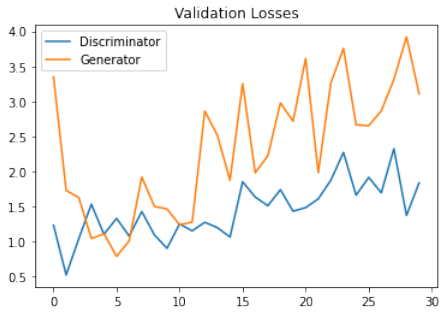

Figure 8: Training Losses of DCGAN

Figure 9: Generated images using DCGAN

4.2.2 WGAN on SVHN

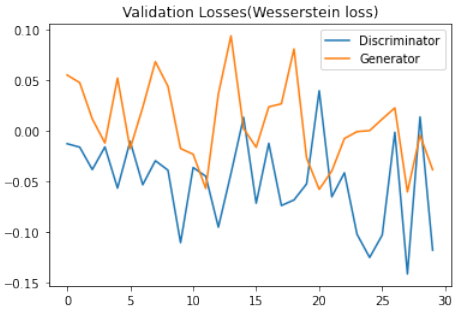

Figure 10: Training Losses of WGAN

Figure 11: Generated images using WGAN

5 Discussion

Due to limited computational power, we trained our network for 30 epochs, except WGAN for MNIST, we trained it for 50000 epochs. In general, if we increase the number of epochs and shuffle the data randomly, the performance of the network will be improved. Also, in convolutional neural networks, we know that different layers can capture different features of the original images, so it may be a good idea to add the output of some hidden layers to the last layer (which can be regarded as a residual structure).

References

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville & Yoshua Bengio. (2014) Generative Adversarial Networks. *arXiv:1406.2661 [stat.ML]*

[2] Emily Denton, Soumith Chintala, Arthur Szlam, & Rob Fergus. (2015) Deep generative image models using a laplacian pyramid of adversarial networks. *arXiv preprint arXiv:1506.05751*

[3] Alec Radford, Luke Metz, & Soumith Chintala. (2015) Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*

[4] Ian J Goodfellow. (2014) On distinguishability criteria for estimating generative models. *arXiv preprint arXiv:1412.6515*

[5] Daniel Jiwoong Im, Chris Dongjoo Kim, Hui Jiang, & Roland Memisevic. (2016) Generating images with recurrent adversarial networks. *arXiv preprint arXiv:1602.05110*

[6] Donggeun Yoo, Namil Kim, Sunggyun Park, Anthony S Paek, & In So Kweon. (2016) Pixel-level domain transfer. *arXiv preprint arXiv:1603.07442*

[7] Martin Arjovsky and Soumith Chintala & Léon Bottou. (2017) Wasserstein GAN. *arXiv preprint arXiv:1701.07875*

[8] Ilya Sutskever, Rafal Jozefowicz, Karol Gregor, et al. (2015) Towards principled unsupervised learning. *arXiv preprint arXiv:1511.06440*

[9] Jost Tobias Springenberg. (2015) Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*

[10] Tensor Flow Tutorial: Deep Convolutional Generative Adversarial Network. (2019) Available at: `https://www.tensorflow.org/tutorials/generative/dcgan` (Accessed: 12 December 2020).

[11] Naoki Shibuya. (2017) Understanding Generative Adversarial Networks [Blog]. Available at: `https://naokishibuya.medium.com/understanding-generative-adversarial-networks-4dafc963f2ef` (Accessed: 12 December 2020).

[12] Jason Brownlee. (2019) A Gentle Introduction to Generative Adversarial Networks (GANs) [Blog]. Available at: `https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/` (Accessed: 12 December 2020).

[13] Vines. (2020) Generative Adversarial Networks - Basically [Blog]. Available at: `https://vinesmsuic.github.io/2020/08/14/gan1/#some-interesting-paper-of-gans` (Accessed: 12 December 2020).

[14] Martin Arjovsky, Léon Bottou. (2017) Towards Principled Methods for Training Generative Adversarial Networks. *arXiv preprint arXiv:1701.04862*